CS5371
Theory of Computation

Lecture 14: Computability V
(Prove by Reduction)

Objectives

- In this lecture, we investigate more undecidable languages
- Instead of proving directly by the diagonalization method, we reduce the problem of deciding A_{TM} to the problem of deciding a language B
- Precisely, we show that if we know how to decide B (i.e., B is decidable), so can A_{TM}. In this way, we show that language B is undecidable

Halting Problem

- Recall that A_{TM} is the language
  { $\langle M, w \rangle$ | M is a TM that accepts w},
  which is undecidable

- Let $HALT_{TM}$ be the language
  { $\langle M, w \rangle$ | M is a TM that halts on w}

Theorem: $HALT_{TM}$ is undecidable

Halting Problem (2)

Proof Idea: Prove by reducing A_{TM} to $HALT_{TM}$. That is, assuming $HALT_{TM}$ is decidable, we then show A_{TM} is decidable.

Let us assume we have a TM R that decides $HALT_{TM}$. (So, what can R do?)

Now, R will accept $\langle M, w \rangle$ if and only if M halts on w. Can we use R to get another TM S that accepts $\langle M, w \rangle$ if and only if M accepts w?

Halting Problem (3)

Proof Idea: Yes! On the input ⟨M, w⟩, the TM S uses TM R to check if M will halt on w. If not, we can immediately reject ⟨M, w⟩ since M does not accept w. (why?)

If yes, we run M on w. The execution must halt, so that there are two cases.

- If M accepts w, S accepts ⟨M, w⟩
- If M rejects w, S rejects ⟨M, w⟩

So, what are the strings that S accepts??

Halting Problem (4)

Let us construct the desired TM S:

S = "On input $\langle M, w \rangle$,

1. Run R on input $\langle M, w \rangle$
2. If R rejects, S rejects
3. If R accepts, simulate M on w
4. If M accepts w, S accepts. Else, S rejects"

Halting Problem (5)

- So, if R is a decider, S is a decider (why?)
- As no decider S can exist, no decider R can exist
- Thus, we conclude that $HALT_{TM}$ is undecidable

Emptiness Test for TM

- Let E_{TM} be the language
  $\{ \langle M \rangle \mid M \text{ is a TM and } L(M) = \{ \} \}$

Theorem: E_{TM} is undecidable

Emptiness Test for TM (2)

Proof Idea: Prove by reducing A_{TM} to E_{TM}. That is, assuming E_{TM} is decidable, we then show A_{TM} is decidable.

Let us assume we have a TM R that decides E_{TM}. (So, what can R do?)

Now, R will accept $\langle M \rangle$ if and only if L(M) is empty. Can we use R to get another TM S that accepts $\langle M, w \rangle$ if and only if M accepts w?

Emptiness Test for TM (3)

Proof Idea: Very tricky…… On the input $\langle M, w \rangle$, we construct another TM M' based on $\langle M, w \rangle$ with the following property:

If M accepts w, $L(M')$ is not empty

If M does not accept w, $L(M')$ is empty

Why do we want to construct such an M'?

To reduce the problem of deciding whether M accepts w to the problem of deciding whether $L(M')$ is empty

Emptiness Test for TM (4)

Proof Idea: Can we find such an M'? Let us find M' with the following property:

If M accepts w, $L(M')$ is $\{w\}$

If M does not accept w, $L(M')$ is empty

Consider the following TM M':

M' = "On input x,

1. If $x \neq w$, reject
2. Run M on x (= w). If M accepts, accept"

Question: What is $L(M')$?

Emptiness Test for TM (5)

Let us construct the desired TM S:

S = "On input $\langle M, w \rangle$,

1. Construct M' based on $\langle M, w \rangle$
2. Run R on $\langle M' \rangle$
3. If R accepts, S rejects $\langle M, w \rangle$ (why?)
4. If R rejects, S accepts $\langle M, w \rangle$"

So, if R is a decider, so is S. (why?) As no decider for S exists, E_{TM} is undecidable

Testing TM with a certain property

Let $REGULAR_{TM}$ be the language

$$\{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$$

Theorem: $REGULAR_{TM}$ is undecidable

Testing TM with a certain property (2)

Proof Idea: Prove by reducing A_{TM} to $REGULAR_{TM}$. That is, assuming $REGULAR_{TM}$ is decidable, we then show A_{TM} is decidable.

Let us assume we have a TM R that decides $REGULAR_{TM}$. (So, what can R do?)

Can we use R to get another TM S that decides $\langle M, w \rangle$?

Testing TM with a certain property (3)

Proof Idea: On the input $\langle M, w \rangle$, we construct another TM M' based on $\langle M, w \rangle$ with the following property:

If M accepts w, $L(M')$ is regular

If M not accept w, $L(M')$ is not regular

Why do we want to construct such an M'?

To reduce the problem of deciding whether M accepts w to the problem of deciding whether $L(M')$ is regular

Testing TM with a certain property (4)

Proof Idea: Can we find such an M'? Let us find M' with the following property:

If M accepts w, $L(M')$ is $\{0,1\}^*$

If M does not accept w, $L(M')$ is $\{0^n1^n\}$

Consider the following TM M':

M' = "On input x,

1. If x has the form 0^n1^n, accept x
2. Else, run M on w. If M accepts, accept x"

Question: What is $L(M')$?

Testing TM with a certain property (5)

Let us construct the desired TM S:

S = "On input $\langle M, w\rangle$,

1. Construct M′ based on $\langle M, w\rangle$
2. Run R on $\langle M'\rangle$
3. If R accepts, S accepts $\langle M, w\rangle$ (why?)
4. If R rejects, S rejects $\langle M, w\rangle$"

So, if R is a decider, so is S. (why?) As no decider for S exists, $REGULAR_{TM}$ is undecidable

Testing TM with a certain property (6)

- We have shown that the language of all TMs having the property "L(M) = regular" is undecidable
- In fact, a general result, called Rice's Theorem, states that the language of all TMs having any specific property is undecidable (Problem 5.28)
- Check this at home!

Equality Test for TM

Let EQ_{TM} be the language

$\{ \langle M_1, M_2 \rangle \mid M_1, M_2$ are TMs, and $L(M_1) = L(M_1) \}$

Theorem: EQ_{TM} is undecidable

Equality Test for TM (2)

Proof Idea: Prove by reducing E_{TM} to EQ_{TM}. That is, assuming EQ_{TM} is decidable, we then show E_{TM} is decidable.

Let us assume we have a TM R that decides EQ_{TM}. (So, what can R do?)

Can we use R to get another TM S that decides if L(M) is empty?

Equality Test for TM (3)

Proof Idea: On the input $\langle M \rangle$, we construct two TMs M_1 and M_2 based on $\langle M \rangle$ with the following property:

If $L(M)$ is empty, $L(M_1) = L(M_2)$

If $L(M)$ not empty, $L(M_1) != L(M_2)$

In this way, we reduce the problem of deciding whether $L(M)$ is empty to the problem of deciding whether the language of two TMs are equal

Equality Test for TM (4)

Proof Idea: Can we find such M_1 and M_2?
Very easy!!! We set M_1 to be M, and M_2 to be a TM that rejects all strings.

Then, M_1 and M_2 has the desired property:
If $L(M)$ is empty, $L(M_1) = L(M_2)$
If $L(M)$ not empty, $L(M_1) = L(M_2)$

Equality Test for TM (5)

Let us construct the desired TM S:

S = "On input ⟨M⟩,

1. Construct M_1 and M_2 based on ⟨M⟩
2. Run R on ⟨M_1, M_2⟩
3. If R accepts, S accepts ⟨M⟩ (why?)
4. If R rejects, S rejects ⟨M⟩"

So, if R is a decider, so is S. (why?) As no decider for S exists, EQ_{TM} is undecidable

Linear Bounded Automaton

Let us now look at languages relating to a new computation model call linear bounded automaton (LBA)

Definition: LBA is a restricted type of TM whose tape head is not allowed to move off the portion of the tape containing the initial input.

Fact: LBA is equivalent to a TM that can use (I.e., read or write) memory of size up to a constant factor of the input length

Linear Bounded Automaton (2)

Theorem: Let M be an LBA with q states and g symbols in the tape alphabet. There are exactly qng^n distinct configurations of M for a tape of length n

Proof: By simple counting… Recall that a configuration specifies the string in the tape (g^n choices in LBA), the position of tape head (n choices in LBA), and the current state (q choices in LBA).

Linear Bounded Automaton (3)

Corollary: On an input of length n, if the LBA M does not halt after qng^n steps, M cannot accept the input

Proof: The computation of M begins with the start configuration. When M performs a step, it goes from one configuration to another. If M does not halt after qng^n steps, some configuration has repeated. Then M will repeat this configuration over and over (why?) ➔ loop

Acceptance by LBA

Let A_{LBA} be the language

$\{\langle M, w\rangle \mid M$ is an LBA and M accepts $w\}$

Theorem: A_{LBA} is decidable

Acceptance by LBA (2)

Proof: Let us construct a decider D:

D = "On input $\langle M, w \rangle$,

1. Simulate M on w for qng^n steps ($n = |w|$) or until it halts
2. If M halts and accepts w, D accepts
3. Else D rejects

Emptiness Test for LBA

Let E_{LBA} be the language

$$\{\langle M\rangle \mid M \text{ is an LBA and } L(M) = \{\ \}\}$$

Theorem: E_{LBA} is undecidable

Emptiness Test for LBA (2)

Proof Idea: Prove by reducing A_{TM} to E_{LBA}. That is, assuming E_{LBA} is decidable, we then show A_{TM} is decidable.

Let us assume we have a TM R that decides E_{LBA}. (So, what can R do?)

Now, R will accept $\langle M \rangle$ if and only if L(M) is empty. Can we use R to get another TM S that accepts $\langle M, w \rangle$ if and only if M accepts w?

Emptiness Test for LBA (3)

Proof Idea: The old idea ...... On the input $\langle M, w\rangle$, we construct an LBA B based on $\langle M, w\rangle$ with the following property:

If M accepts w, $L(B)$ is not empty

If M does not accept w, $L(B)$ is empty

Why do we want to construct such an B?

To reduce the problem of deciding whether B accepts w to the problem of deciding whether $L(B)$ is empty

Emptiness Test for LBA (4)

Proof Idea: … Before we do so, let us recall that an accepting configuration of a TM is a configuration whose current state is q_{accept}

We define an accepting computation history to be a finite sequence of configurations $C_0, C_1, \ldots, C_k$ such that C_0 is the start configuration, each C_i follows legally from C_{i-1}, and finally C_k is an accepting configuration

Emptiness Test for LBA (5)

Proof Idea: That means, whenever $\langle M, w \rangle$ is in A_{TM}, there must be an accepting configuration history that M goes through as it accepts w

Back to our proof…

We shall construct LBA B to accept one string whenever M accepts w, and accepts nothing whenever M does not accept w

(Guess: what is this special string?)

Emptiness Test for LBA (6)

Proof Idea: The special string is the accepting computation history:

$\# \; C_0 \; \# \; C_1 \; \# \; C_2 \; \# \; \ldots \; \# \; C_k \; \#$

The construction of B is easy:

B = "On input x,

1. Test if x is an accepting computation history for M to accept w
2. If yes, accept x
3. Else rejects

Emptiness Test for LBA (7)

Quick Quiz:

Q1: Can B be constructed in finite steps?

Q2: What is L(B)?

Q3: Is B an LBA?

Emptiness Test for LBA (8)

Let us construct the desired TM S for A_{TM}:

S = "On input $\langle M, w \rangle$,

1. Construct LBA B based on $\langle M, w \rangle$
2. Run R (LBA emptiness-tester) on $\langle B \rangle$
3. If R accepts, S rejects $\langle M, w \rangle$ (why?)
4. If R rejects, S accepts $\langle M, w \rangle$"

So, if R is a decider, so is S. (why?) As no decider for S exists, E_{LBA} is undecidable

CFG Accepting All Strings

Let ALL_{CFG} be the language

$$\{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^* \}$$

Theorem: ALL_{CFG} is undecidable

CFG Accepting All Strings (2)

Proof Idea: Prove by reducing A_{TM} to ALL_{CFG}. That is, assuming ALL_{CFG} is decidable, we then show A_{TM} is decidable.

Let us assume we have a TM R that decides ALL_{CFG}. (So, what can R do?)

Now, R will accept $\langle G \rangle$ if and only if L(G) accepts all strings. Can we use R to get another TM S that accepts $\langle M, w \rangle$ if and only if M accepts w?

CFG Accepting All Strings (3)

Proof Idea: The old idea …… On the input $\langle M, w \rangle$, we construct an CFG G based on $\langle M, w \rangle$ with the following property:

If M accepts w, $L(G)$ is not all strings

If M does not accept w, $L(G)$ is all strings

(Guess: what are the missing strings when M accepts w?)

If M accepts w, we want $L(G)$ contains all but any accepting computation histories for M to accept w

CFG Accepting All Strings (4)

Proof Idea: How can we find this grammar G? Very tricky, but here is one way:

Let G generates all strings that:

1. Do not start with C_0 (Note: C_0 is based on M,w)
2. Do not end with an accepting configuration
3. Some C_i does not follow legally from C_{i-1}

CFG Accepting All Strings (5)

Quick Quiz:

Q1: Does such a CFG G exist?

Q2: Can G be constructed in finite steps?

Q3: What is $L(G)$?

$L(G)$ = all but accepting if M accepts w

$L(G)$ = all strings if M does not accept w

CFG Accepting All Strings (6)

Let us construct the desired TM S for A_{TM}:

S = "On input $\langle M, w \rangle$,

1. Construct CFG G based on $\langle M, w \rangle$
2. Run R (all-CFG-tester) on $\langle G \rangle$
3. If R accepts, S rejects $\langle M, w \rangle$ (why?)
4. If R rejects, S accepts $\langle M, w \rangle$"

So, if R is a decider, so is S. (why?) As no decider for S exists, ALL_{CFG} is undecidable

Next Time

- Post's Correspondence Problem
  - An undecidable problem with dominos
- Computable functions
  - Another way of looking at reduction